###########################################################################################################

############################ R Code- Part1. R function for Simulation Study ########################################

###########################################################################################################

## PAPER: Local community assembly mechanisms shape soil bacterial beta-diversity patterns along latitudinal

## gradients in eastern China

## JOURNAL: (to be submitted to Nature Communications)

## AUTHORS: Xiao Zhang Shirong Liu\*, Jingxin Wang, Yongtao Huang, Zachary Freedman, Shenglei Fu, Hui Wang,

## Xiangzhen Li, Minjie Yao, Xiaojing Liu, Kuan Liu, Jamie Schuler

##

## Code last update on April 15st, 2020

##

## DESCRIPTION: This is a modified version of the scripts Kraft NJ, et al.Science 333, 1755-1758 (2011),

## dependencies found here: https://science.sciencemag.org/content/suppl/2011/09/21/333.6050.1755.DC1

############ R code for simulation study with a lognormal species abundance distribution ##################

#Generating a vector with 10000 elements on regional gamma diversity;

#Gamma diversity range from 1 to 10000, with 200 increments between settings;

gamma\_vect <- seq(1,10000, by = 200)

# set the number of plots per region(ncom) and individuals per plot (num\_individuals)

ncom <- 60

num\_individuals <- c(500, 1500, 3000, 5500, 10000)

# This function is used to generate simulated community given gamma diversity and number of individuals per plot;

sim\_com <- function(gamma\_d, alpha\_d)

{

# hypothesize lognormal species abundance distribution

# abund\_pool~LogNorm(mean = mean\_abund, sd = sd\_abund)

#Setting lognormal distribution parameters

mean\_abund <- alpha\_d/gamma\_d

# the variable coefficient setup 1

cv\_abund <- 1

sd\_abund <- mean\_abund \* cv\_abund #given cv\_abund=1, sd\_abund=mean\_abund;

# Given log(abund\_pool)~Norm(mean=mu1, sd=sigma1),

# calculating the corresponding normal distribution parameters;

sigma1 <- sqrt(log(sd\_abund^2/mean\_abund^2 + 1))

mu1 <- log(mean\_abund) - sigma1^2/2

# simulating species abundance distribution from lognormal distribution;

abund\_pool <- rlnorm(n = gamma\_d, meanlog = mu1, sdlog = sigma1)

# calulating simulated relative abundance;

# ranking and sorting from largest relative abundance to the smallest;

rel\_abund\_pool <- abund\_pool/sum(abund\_pool)

rel\_abund\_pool <- sort(rel\_abund\_pool, decreasing = T)

names(rel\_abund\_pool) <- paste0("species",seq\_along(rel\_abund\_pool))

# generate the simulated community from the pool via random sampling with replacement;

# relative abundance is used to weight the probability on sample selection;

sample\_vec <- sample(x = names(rel\_abund\_pool), size = alpha\_d,

replace = TRUE, prob = rel\_abund\_pool)

sample\_vec <- factor(sample\_vec, levels = names(rel\_abund\_pool))

abund\_local <- table(sample\_vec)

class(abund\_local) <- c("integer")

abund\_local #the function will return a vector on local abundance;

}

set.seed(1)

library(vegan)

# create a matrix to store all the simulated results

beta\_obs\_matrix\_mean <- matrix(0,nrow=length(num\_individuals), ncol=length(gamma\_vect))

for (i in 1:length(num\_individuals)) {

beta\_obs\_mean <- NULL

for(j in 1:length(gamma\_vect)){

# obtain the simulated communities

simulated\_communities = NULL

k = 0

while(k < ncom){

simulated\_communities <- rbind(simulated\_communities, sim\_com(gamma\_d = gamma\_vect[j], alpha\_d = num\_individuals[i]))

k <- k + 1

}

# calculate beta-diversity as the distance (or compositional dissimilarity) from an individual

# plot to the centroid of the group of all plots within a region (distance-to-centroid) using the multivariate method

# based on Bray-Curtis metric

beta\_partition <- vegdist(simulated\_communities, method="bray")

if(mean(beta\_partition) == 0){

beta\_partition.distance <- 0

}else{

beta\_partition.bdisper <- betadisper(beta\_partition, group=rep(1, times=nrow(simulated\_communities)),type="centroid")

beta\_partition.distance <- mean(beta\_partition.bdisper$distances)

}

beta\_obs\_mean <- c(beta\_obs\_mean, beta\_partition.distance)

}

beta\_obs\_matrix\_mean[i,] <- beta\_obs\_mean

}

par(mgp=c(1.9,0.5,0))

plot(gamma\_vect, beta\_obs\_matrix\_mean[1,], type = "l", ylim = c(0.01,0.69),

xlim=c(480,9400),xlab="gamma diversity", ylab="beta diversity",

cex.lab=1.25,tck=-0.02, lwd=2)

for (m in 2:length(num\_individuals)) {

points(gamma\_vect, beta\_obs\_matrix\_mean[m,], type = "l", lwd=2)

}

############# R code for simulation study with a uniform species abundance distribution #############

#Generating a vector with 10000 elements on regional gamma diversity;

#Diversity range from 1 to 10000, with 200 increments between settings;

gamma\_vect <- seq(1,10000, by = 200)

# set the number of plots per region (ncom) and individuals per plot (num\_individuals)

ncom <- 60

num\_individuals <- c(500, 1500, 3000, 5500, 10000)

# This function is used to generate simulated community given gamma diversity and number of individuals per plot;

sim\_com <- function(gamma\_d, alpha\_d)

{

# simulating species abundance distribution from uniform distribution

rel\_abund\_pool <- rep(1/gamma\_d, gamma\_d)

names(rel\_abund\_pool) <- paste0("species",seq\_along(rel\_abund\_pool))

# generate the simulated community from the pool via random sampling with replacement;

# relative abundance is used to weight the probability on sample selection;

sample\_vec <- sample(x = names(rel\_abund\_pool), size = alpha\_d,

replace = TRUE, prob = rel\_abund\_pool)

sample\_vec <- factor(sample\_vec, levels = names(rel\_abund\_pool))

abund\_local <- table(sample\_vec)

class(abund\_local) <- c("integer")

abund\_local #the function will return a vector on local abundance;

}

set.seed(1)

library(vegan)

# create a matrix to store all the simulated results

beta\_obs\_matrix\_mean <- matrix(0,nrow=length(num\_individuals), ncol=length(gamma\_vect))

for (i in 1:length(num\_individuals)) {

beta\_obs\_mean <- NULL

for(j in 1:length(gamma\_vect)){

# obtain the simulated communities

simulated\_communities = NULL

k = 0

while(k < ncom){

simulated\_communities <- rbind(simulated\_communities, sim\_com(gamma\_d = gamma\_vect[j], alpha\_d = num\_individuals[i]))

k <- k + 1

}

# calculate beta-diversity as the distance (or compositional dissimilarity) from an individual

# plot to the centroid of the group of all plots within a region (distance-to-centroid) using the multivariate method

# based on Bray-Curtis metric

beta\_partition <- vegdist(simulated\_communities, method="bray")

if(mean(beta\_partition) == 0){

beta\_partition.distance <- 0

}else{

beta\_partition.bdisper <- betadisper(beta\_partition, group=rep(1, times=nrow(simulated\_communities)),type="centroid")

beta\_partition.distance <- mean(beta\_partition.bdisper$distances)

}

beta\_obs\_mean <- c(beta\_obs\_mean, beta\_partition.distance)

}

beta\_obs\_matrix\_mean[i,] <- beta\_obs\_mean

}

par(mgp=c(1.9,0.5,0))

plot(gamma\_vect, beta\_obs\_matrix\_mean[1,], type = "l", ylim = c(0.01,0.69),

xlim=c(480,9400),xlab="gamma diversity", ylab="beta diversity",

cex.lab=1.25,tck=-0.02, lwd=2)

for (m in 2:length(num\_individuals)) {

points(gamma\_vect, beta\_obs\_matrix\_mean[m,], type = "l", lwd=2)

}

###################################################################################################

############################## R Code- Part2. R Functions for Null Model Analyses ###########################

###################################################################################################

## R code from Reference article, Vannette R L, et al.Ecology Letters, 2017, 20(7): 901-910,

## which is avaliable at??https://onlinelibrary.wiley.com/doi/full/10.1111/ele.12787?scrollTo=references

## Consistent with Vannette R L, et al. 2017, code also depends on functions defined by Tello et al. 2015,

## PLOS ONE, 2015, 10(3), which were found here:

## http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0121458#sec014

##

##

## DESCRIPTION: The following two functions were used for null-model analyses.

## 1. 'assemblages.from.pool.randA' was used to produce null local assemblages expected by random

## sampling from observed species pools while eliminating the effect of mechanisms controlling for

## either one or both the two components of regional species pool (gamma diversity and the regional

## species abundance). fix.rSAD - TRUE/FALSE argument that determines whether the regional species

## abundance distribution (SAD; column sums) will be constrained to be the same as in the empirical data.

## If FALSE,individuals are randomly re-assigned to species before they are re-distributed among local

## assemblages. In this way, the species pool is defined only as the observed number of species in a

## region (gamma diversity).If TRUE,the regional species abundance distribution is constrained to be

## the same in null and empirical datasets.Here,the species pool is defined as the observed number

## of species and abundances of species in a region.

##

## 2. 'beta\_ses' was used to calculate beta-diversity from empirical and null assemblages,

## that is, beta-deviations.

############################# FUNCTION 1 - assemblages.from.pool.randA ###########################

# SUMMARY

# A randomization algorithm that produces matrices of species composition expected by randomly

# re-assigning individuals from the species pool into local assemblages.

# USAGE

# assemblages.from.pool.randA(compo, rand.N=999, fix.local.abund=TRUE, fix.rSAD=TRUE,

# save.output=FALSE, save.format="matrices", path.to.save, show.progress=FALSE)

# ARGUMENTS

# compo - a matrix or data frame where rows are sites and columns species. Values within the

# matrix represent abundances as integers. The function will not work with presence/absence

# matrices or measures of abundance other than number of individuals.

#

# rand.N - an integer determining the number of null matrices to be produced.

#

#

# fix.rSAD - TRUE/FALSE argument that determines whether the regional species abundance distribution

# (SAD; column sums) will be constrained to be the same as in the empirical data. If FALSE,

# individuals are randomly re-assigned to species before they are re-distributed among local

# assemblages. The default is TRUE.

#

# fix.local.abund - TRUE/FALSE argument that determines whether the abundances in local assemblages

# (row sums) will be constrained to be the same as in the empirical data. The default is

# TRUE.(In our study, we disentangled community assembly processes based on both 'fix.local.abund = TRUE'

# and 'fix.local.abund = FALSE'. The two results were almost consistent and the former was shown in the study.)

#

# save.output - TRUE/FALSE argument that defines whether results should be saved into files

# (when TRUE), or should be returned into the console (when FALSE). The default is FALSE.

#

# save.format - character argument that can take one of two values: "matrices" or "list". This

# argument applies only if 'save.output' is TRUE. When the "matrices" option is requested, each

# of the 'rand.N' null matrices will be written into a file as the function finishes each

# randomization. If the "list" option is requested, a single list containing all 'rand.N' null

# matrices will be saved after all randomizations have been completed. The default is "matrices".

#

# path.to.save - character argument that gives the address of the folder in the local computer

# where output files should be saved. This argument applies only if 'save.output' is TRUE.

# The default is the working directory. The path should end with the name of the target folder,

# not with '\\' or '/'.

#

# show.progress - TRUE/FALSE argument that defines whether the progress of the function across

# randomizations should be shown in the console. The default is FALSE.

# OUTPUT

# A list with the following 2 elements:

#

# rand.parameters - a vector containing information on the parameters used for the randomization of

# the data.

#

# rand.datasets - When 'save.output' is FALSE, this is a list of length 'rand.N'. Each element of

# the list contains a null matrix of species composition that is of identical dimensions as the

# empirical matrix ('compo'). When 'save.output' is TRUE, 'rand.datasets' is simply the path given

# by the argument 'path.to.save', and the composition matrices are saved into either a single

# list file (when 'save.format' is "list"), or to individual files for each matrix (when

# 'save.format' is "matrices").

assemblages.from.pool.randA <- function(compo, rand.N=999, fix.local.abund=TRUE, fix.rSAD=TRUE,

save.output=FALSE, save.format="matrices", path.to.save, show.progress=FALSE)

{

## Makes sure there is a path to save files if needed

if(save.output==TRUE & missing(path.to.save)) path.to.save <- getwd()

## Makes sure 'compo' is in the right format

compo <- as.matrix(compo, ncol=ncol(compo))

if(is.null(rownames(compo))) rownames(compo) <- paste("site\_", 1:nrow(compoT), sep="")

if(is.null(colnames(compo))) colnames(compo) <- paste("sp\_", 1:nrow(compoT), sep="")

## Calculates density of individuals per site

site.densities <- rowSums(compo)

if(min(site.densities)<=0) warning("Some sites have no species")

## Calculates total regional abundance

regional.abundance <- sum(site.densities)

## Calculates regional abundances per species - the regional SAD

spp.abund <- colSums(compo)

## Checks for a couple of potential problems

if(length(which((compo - round(compo, 0)) != 0)) > 0)

stop("This function can not randomize a matrix with species abundances that are not integers")

if(min(spp.abund)<=0)

warning("Some species have abundances of less than 1")

## Finds the species with at least one individual

spp.more.than.0.indices <- which(spp.abund>0)

spp.more.than.0.names <- colnames(compo)[spp.more.than.0.indices]

## Produces a vector of species identities for each individual

individual.id <- as.vector(rep(colnames(compo), spp.abund))

## Produces an empty list to save null composition matrices

rand.datasets <- sapply(rep(NA, rand.N), list)

names(rand.datasets) <- paste("RandDataset\_", 1:rand.N, sep="")

for(i in 1:rand.N)

{

if(show.progress==TRUE)

print(i)

## When the regional SAD is NOT fixed, produces a null SAD to be used in analyses by

## randomly assigning individuals to species

if(fix.rSAD==FALSE)

{

# Assigns individuals to species at random

null.sp.to.ind.assignation.1 <- spp.more.than.0.names # IMPORTANT - This ensures that all

# species in the region receive at least

# one individual in the null SAD

null.sp.to.ind.assignation.2 <- character()

if((regional.abundance-length(spp.more.than.0.names))>0)

null.sp.to.ind.assignation.2 <- sample(spp.more.than.0.names,

regional.abundance-length(spp.more.than.0.names), replace=TRUE)

null.sp.to.ind.assignation <- c(null.sp.to.ind.assignation.1, null.sp.to.ind.assignation.2)

# calculates null abundances for species present with at least one individual

pre.null.spp.abund <- as.numeric(table(null.sp.to.ind.assignation))

pre.null.spp.abund <- pre.null.spp.abund[sample(1:length(pre.null.spp.abund))]

# Inserts the null abundances into a vector that might contain some zeroes (i.e., if

# there is species present in the empirical composition table that do not have any

# individuals)

null.spp.abund <- spp.abund

null.spp.abund[spp.more.than.0.indices] <- pre.null.spp.abund

# Produces a vector of null species identities for each individual

null.individual.id <- as.vector(rep(colnames(compo), null.spp.abund))

}

## When the regional SAD is fixed, makes the null SAD identical to the empirical SAD

if(fix.rSAD==TRUE)

{

null.spp.abund <- spp.abund

null.individual.id <- individual.id

}

## Assigns individuals to local sites at random

if(fix.local.abund==TRUE)

null.site.assignation <- sample(rep(rownames(compo), times=site.densities),

regional.abundance, replace=FALSE)

if(fix.local.abund==FALSE)

null.site.assignation <- sample(rownames(compo), regional.abundance, replace=TRUE)

## Creates a null species composition matrix using the null assignation of individuals

## to sites

null.compo <- tapply(rep(1, regional.abundance), list(null.site.assignation,

null.individual.id), sum)

null.compo[is.na(null.compo)] <- 0

## Reshapes the null composition matrix in case there is only 1 species

if(ncol(compo)==1)

{

null.compo <- matrix(null.compo, ncol=1)

colnames(null.compo) <- colnames(compo)

rownames(null.compo) <- unique(null.site.assignation)

}

## Inserts empty plots into the null composition matrix in case there are any

if(nrow(null.compo)<nrow(compo))

{

missing.plots <- setdiff(rownames(compo), rownames(null.compo))

empty.plots <- matrix(0, ncol=ncol(null.compo), nrow=length(missing.plots))

colnames(empty.plots) <- colnames(null.compo)

rownames(empty.plots) <- missing.plots

null.compo <- rbind(null.compo, empty.plots)

if(ncol(compo)==1)

{

null.compo <- matrix(null.compo, ncol=1)

colnames(null.compo) <- colnames(compo)

rownames(null.compo) <- c(unique(null.site.assignation), missing.plots)

}

}

## Matches the column and row names in the null and empirical composition matrices

if(ncol(compo)>1) null.compo <- null.compo[,match(colnames(compo), colnames(null.compo))]

null.compo <- null.compo[match(rownames(compo), rownames(null.compo)),]

## Reshapes the null composition matrix in case there is only 1 species

if(ncol(compo)==1)

{

null.compo <- matrix(null.compo, ncol=1)

colnames(null.compo) <- colnames(compo)

rownames(null.compo) <- rownames(compo)

}

## Adds names to the species with zero abundances (i.e., empty columns)

if(min(null.spp.abund)<=0)

{

colnames(null.compo)[null.spp.abund<=0] <- colnames(compo)[null.spp.abund<=0]

null.compo[is.na(null.compo)] <- 0

}

## Adds the null composition matrix to the list that compiles the results

if(save.output==FALSE | save.format=="list")

rand.datasets[[i]] <- null.compo

## If requested, the null composition matrix is saved as a file

if(save.output==TRUE & save.format=="matrices")

write.table(null.compo, file=paste(path.to.save, "\\", "RandDataset\_", i, ".txt", sep=""),

quote=FALSE, sep="\t", na="NA", dec=".", row.names=TRUE, col.names=TRUE)

}

## If saving a list is requested, the full list of null composition matrices is saved as a file

if(save.format=="list")

save(rand.datasets, file=paste(path.to.save, "\\", "RandDatasets", sep=""))

## Makes a list of the parameters used in the randomization

rand.parameters <- c(fix.local.abund, fix.rSAD, rand.N)

names(rand.parameters) <- c("fix.local.abund", "fix.rSAD", "rand.N")

## Creates the output to return, depending on whether the main output was saved or not

if(save.output==TRUE)

output <- list(rand.parameters, path.to.save)

else

output <- list(rand.parameters, rand.datasets)

names(output) <- c("rand.parameters", "rand.datasets")

output

}

############################# FUNCTION 2 - code for the function beta.ses (beta-deviation) ###########################

beta.ses <-function(compo, null.matrices)

{

## Opens required packages

require(vegetarian)

require(vegan)

## Makes sure 'compo' is in the right format

dim.compo <- length(dim(compo))

if(dim.compo==0)

stop("'compo' does not have more than 1 dimension")

if(dim.compo>2)

stop("This function does not know what to do when 'compo' has more than 2 dimensions")

if(dim.compo==2)

{

if(is.null(rownames(compo)))

rownames(compo) <- paste("site\_", 1:nrow(compo), sep="")

if(is.null(colnames(compo)))

colnames(compo) <- paste("sp\_", 1:nrow(compo), sep="")

}

## Checks that values in the cells are all positive integers

if(sum((compo - round(compo, 0)) != 0) > 0)

stop("This function cannot use species abundances that are not integers")

if(sum(compo<0) > 0)

stop("This function cannot use negative species abundances")

if(sum(is.na(compo)) > 0)

stop("This function cannot use NAs")

fit <- vegdist(compo, method="bray")

emp.bdisp.dist <- betadisper(fit, group=rep(1, times=nrow(compo)),type="centroid")

emp.dist <- emp.bdisp.dist$distances

## Calculates density of individuals per site

site.densities <- rowSums(compo)

if(min(site.densities)==0)

warning("Some empirical sites seem to be empty of individuals")

## Calculates regional abundances per species - the regional SAD

spp.abund <- colSums(compo)

if(min(spp.abund)<=0)

warning("Some species have abundances of less than or equal to zero")

## Calculates species richness across all sites

regional.richness <- length(which(spp.abund>0))

## Calculates species richness at each site

site.richness <- rowSums(compo>0)

mean.site.richness <- mean(site.richness, na.rm=TRUE)

## Calculates the number of null matrices to use

null.N <- length(null.matrices)

## Creates empty objects to hold results from the null matrices

## need a matrix length (empirical) x width (# of null communities)

rand.distances <- as.data.frame(matrix(NA, nrow=nrow(compo), ncol=null.N))

rownames(rand.distances) <- rownames(compo)

colnames(rand.distances) <- paste("null\_", 1:null.N, sep="")

for (i in 1:null.N)

{

spp.abund <- colSums(compo)

## Defines the focal null composition matrix

null.compo <- as.matrix(null.matrices[[i]])

## Calculates null regional abundances per species - the regional SAD

null.spp.abund <- colSums(null.compo)

if(identical(null.spp.abund, spp.abund)==FALSE)

warning("Simulated and observed species total abundances are not identical")

## Calculates species richness across all sites

null.regional.richness <- length(which(null.spp.abund>0))

## Calculates species richness at each site

null.site.richness <- rowSums(null.compo>0)

null.mean.site.richness <- mean(null.site.richness)

## Calculates null composition distances among all possible pairs of sites

null.dist.details <- vegdist(null.compo, method="bray")

rand.bdisper <- betadisper(null.dist.details, group=rep(1, times=nrow(null.compo)),type="centroid")

rand.distances[,i] <- rand.bdisper$distances

}

# now use this to calculate the ses for each value

SES <- as.data.frame(matrix(NA, nrow=nrow(compo)), ncol=1)

ES <- (emp.dist-rowMeans(rand.distances))

for(i in 1:length(ES)){

SES[i,1]<- ES[i]/sd(rand.distances[i,])

}

SES

}

############################# Example usage of beta ses function ######################################

## dat - a matrix or data frame where rows are sites and columns species. Values within the

## matrix represent abundances as integers. The function will not work with presence/absence

## matrices or measures of abundance other than number of individuals.

rand.results <- assemblages.from.pool.randA(compo=t(otu\_table(dat)), fix.local.abund=TRUE, fix.rSAD=TRUE,rand.N=N, save.output=FALSE)

null.compos <- rand.results$rand.datasets

null.matrices <- null.compos

compo<- t(otu\_table(dat))

beta\_deviation <- beta.ses(compo, null.matrices)

###################################### SUPPLEMENTARY FUNCTIONS ####################################

############## Evaluating the difference between observed β-diversity and expected β-diversities##################

####################################################################################################

### SUPPLEMENTARY FUNCTION 1: summary.rand #########################################################

summary.rand <- function(empirical, rand.details, critial.quantiles=c(0.025, 0.975), boot.details,

boot.CI.quantiles=c(0.025, 0.975))

{

emp.names <- rownames(as.matrix(empirical))

rand.names <- colnames(rand.details)

if(length(emp.names)==0 | length(rand.names)==0)

stop("Both 'empirical' and 'rand.details' must have 'names' or 'colnames'")

if(identical(emp.names, rand.names)==FALSE)

stop("Names in 'empirical' do not match names in 'rand.details'")

if(missing(boot.details)==FALSE)

{

boot.names <- colnames(boot.details)

if(length(boot.names)==0)

stop("Both 'empirical' and 'rand.details' must have 'names' and 'colnames'")

if(identical(emp.names, boot.names)==FALSE)

stop("Names in 'empirical' do not match names in 'boot.details'")

}

empirical <- as.numeric(empirical)

rand.details <- as.matrix(rand.details)

if(length(empirical)!=ncol(rand.details))

stop("Number of empirical values and of columns in randomization matrix do not match")

rand.N <- apply(!is.na(rand.details), 2, sum)

NA.rand.N <- apply(is.na(rand.details), 2, sum)

mean.rand <- colMeans(rand.details, na.rm=TRUE)

median.rand <- apply(rand.details, 2, median, na.rm=TRUE)

sd.rand <- apply(rand.details, 2, sd, na.rm=TRUE)

critical.values <- as.matrix(t(apply(rand.details, 2, quantile, probs=critial.quantiles,

na.rm=TRUE)))

ps <- t(sapply(1:length(empirical), rand.p, empirical=empirical, rand.details=rand.details))

ES <- empirical - mean.rand

SES <- ES/sd.rand

which.inf <- which(sd.rand==0)

if(length(which.inf)>0)

{

warning("At least some statistics showed no variation expected by the randomization

alrorithm - standard deviations of randomized values equal to zero. Corresponding SES

values were set to NA", call.=FALSE)

SES[which.inf] <- NA

}

summary.results <- data.frame(empirical, rand.N, NA.rand.N, mean.rand, median.rand, sd.rand, ES,

SES, critical.values, ps)

colnames(summary.results) <- c("empirical", "rand.N", "NA.rand.N", "mean.rand", "median.rand",

"sd.rand", "ES", "SES", "lower.critical.value", "upper.critical.value", "p.less.than",

"p.greater.than")

rownames(summary.results) <- emp.names

if(missing(boot.details)==FALSE)

{

mean.boot <- colMeans(boot.details, na.rm=TRUE)

CI.boot <- as.matrix(t(apply(boot.details, 2, quantile, probs=boot.CI.quantiles, na.rm=TRUE)))

colnames(CI.boot) <- c("lower.CI.boot", "upper.CI.boot")

summary.results <- data.frame(summary.results, mean.boot, CI.boot)

}

summary.results

}

################################### SUPPLEMENTARY FUNCTION 2: rand.p ######################################

rand.p <- function(i, empirical, rand.details)

{

emp.i <- empirical[i]

rands.i <- rand.details[,i]

rands.i <- rands.i[which(is.na(rands.i)==FALSE)]

p.less.than <- length(which(rands.i <= emp.i)) / length(rands.i)

p.greater.than <- length(which(rands.i >= emp.i)) / length(rands.i)

rand.ps <- cbind(p.less.than, p.greater.than)

names(rand.ps) <- c("p.less.than", "p.greater.than")

rand.ps

}

###################################################################################################

############################## R Code- Part3. R Functions for PCNM, STEP and VPA ##########################

###################################################################################################

## Function to partition variation in the beta-deviations into individual fractions explained by environmental and spatial variables

## The function takes the data for a region that contains beta deviations, environmental and spatial variables.

## see example data file: "lat\_lon.csv, beta\_dev.csv, spa.csv, env.csv"

library(SoDA)

library(AEM)

library(packfor)

lat <-read.csv("lat\_lon.csv",row.names=1)

dev <- read.csv("beta\_devi.csv", row.names=1)

spa <- read.csv("spa.csv", row.names=1)

env <- read.csv("env.csv", row.names=1)

## canculate Geodetic coordinates from latitude and longitude

xy <- geoXY(lat[,1], lat[,2], unit = 1000)

## computed classical Principal Coordinates of Neighbourhood Matrix (PCNM). These are used to

## transform (spatial) distances to rectangular data that suitable for constrained ordination or regression.

'PCNM' <- function(matdist, thresh=NULL, dbMEM=FALSE, moran=NULL, all=FALSE, include.zero=FALSE, silent=FALSE)

#

# Compute the PCNM or dbMEM eigenfunctions corresponding to

# all eigenvalues (+, 0, -).

# In PCNM computation, the diagonal of D = 0.

# In dbMEM, the diagonal of D = 4\*threshh.

# Distance-based MEM are described in Dray et al. 2006.

# The name was abbreviated to db-MEM by PPN & PL (subm.)

# Input file: distance matrix produced by the function "dist".

# Computation of the threshold requires a function of the library "ape".

#

# Original PCNM function: Stephane Dray, November 11, 2004

# The present version: Pierre Legendre, August 2007, January and March 2009

{

require(vegan)

epsilon <- sqrt(.Machine$double.eps)

a <- system.time({

if(is.null(moran)) {

if(dbMEM) { moran=FALSE } else { moran=TRUE }

}

single <- FALSE

if(moran) {

# cat("The site coordinates were computed from 'matdist'.",'\n')

pcoa.xy <- pcoa.all(matdist)

if(is.na(pcoa.xy$values[2]) | (pcoa.xy$values[2] < epsilon)) {

if(!silent) cat("The sites form a straight line on the map.",'\n')

xy <- pcoa.xy$vectors

single <- TRUE

} else {

xy <- pcoa.xy$vectors[,1:2]

}

}

matdist <- as.matrix(matdist)

n <- nrow(matdist)

# Truncation of distance matrix

if(is.null(thresh)) {

spanning <- vegan::spantree(as.dist(matdist))

threshh <- max(spanning$dist)

if(!silent) cat("Truncation level =",threshh+0.000001,'\n')

} else {

threshh = thresh

if(!silent) cat("User-provided truncation threshold =",thresh,'\n')

}

matdist[matdist > threshh] <- 4\*threshh

if(dbMEM==FALSE) { diagonal <- 0 } else { diagonal <- 4\*threshh }

mypcnm.all <- pcoa.all(matdist, diagonal=diagonal, all=all, include.zero=include.zero, rn=rownames(matdist))

# Compute Moran's I

if(moran) {

require(AEM)

if(single) {

nb <- dnearneigh(matrix(c(xy,rep(0,n)),n,2), 0, (threshh + epsilon))

} else {

nb <- dnearneigh(xy, 0, (threshh + epsilon))

}

fr.to.pcnm2 <- as.matrix(listw2sn(nb2listw(nb))[,1:2])

weight.dist.coord.mat <- as.matrix(1-(as.dist(matdist)/(4\*threshh))^2)

weight <- weight.dist.coord.mat[fr.to.pcnm2]

res <- moran.I.multi(mypcnm.all$vectors, link=fr.to.pcnm2, weight=weight)

Moran <- res$res.mat[,1:2]

positive <- rep(FALSE,length(mypcnm.all$values))

positive[which(Moran[,1] > res$expected)] <- TRUE

Moran <- cbind(as.data.frame(Moran), positive)

colnames(Moran) <- c("Moran","p.value","Positive")

}

})

a[3] <- sprintf("%2f",a[3])

if(!silent) cat("Time to compute PCNMs =",a[3]," sec",'\n')

if(is.null(thresh)) {

if(moran) {

res <- list(values=mypcnm.all$values, vectors=mypcnm.all$vectors, Moran\_I=Moran, expected\_Moran=res$expected, spanning=spanning, thresh=threshh+0.000001)

} else {

res <- list(values=mypcnm.all$values, vectors=mypcnm.all$vectors, spanning=spanning, thresh=threshh+0.000001)

}

} else {

if(moran) {

res <- list(values=mypcnm.all$values, vectors=mypcnm.all$vectors, Moran\_I=Moran, expected\_Moran=res$expected, thresh=thresh)

} else {

res <- list(values=mypcnm.all$values, vectors=mypcnm.all$vectors, thresh=threshh+0.000001)

}

}

res

}

'pcoa.all' <- function(D, diagonal=0, all=FALSE, include.zero=FALSE, rn=NULL)

# Principal coordinate decomposition of a square distance matrix D

# Get the eigenvectors corresponding to all eigenvalues, positive and negative

# Pierre Legendre, 2005, 2007

#

# D : A distance matrix of class 'dist' or 'matrix'.

# all : If TRUE, the eigenvectors corresponding to all eigenvalues, positive and negative, are shown in the output list.

# include.zero : If FALSE (default value), the zero eigenvalues as well as their eigenvectors are excluded from the output list.

# rn : An optional vector of row names, of length n, for the objects.

{

epsilon <- sqrt(.Machine$double.eps)

# replace by: epsilon <- .Machine$double.eps \* 10^2

D <- as.matrix(D)

n <- nrow(D)

D <- D + diag(rep(diagonal,n))

# Gower centring, matrix formula

One <- matrix(1,n,n)

mat <- diag(n) - One/n

Dpr2 <- -0.5 \* mat %\*% (D^2) %\*% mat

trace <- sum(diag(Dpr2))

# Eigenvalue decomposition

D.eig <- eigen(Dpr2, symmetric=TRUE)

rel.values <- D.eig$values/trace

rel.cum <- cumsum(rel.values)

if(length(rn)!=0) {

rownames(D.eig$vectors) <- rn

} else {

rownames(D.eig$vectors) <- rownames(D)

}

# Output the results: k eigenvalues and eigenvectors

if(all) {

select <- 1:n

if(!include.zero) {

exclude <- which(abs(D.eig$values) < epsilon)

select <- select[-exclude]

}

k <- length(select)

res <- list(values=D.eig$values[select], rel.values=rel.values[select], rel.cum.values=rel.cum[select], vectors=D.eig$vectors[,select], trace=trace)

# cat("k =",k,"Select =",select,'\n')

} else {

k <- length(which(D.eig$values > epsilon))

weight <- sqrt(D.eig$values[1:k])

if(k == 1) {

vectors <- D.eig$vectors[,1]\*sqrt(D.eig$values[1])

} else {

vectors <- D.eig$vectors[,1:k]%\*%diag(weight)

}

res <- list(values=D.eig$values[1:k], rel.values=rel.values[1:k], rel.cum.values=rel.cum[1:k], vectors=vectors, trace=trace)

}

res

}

PCNM <- PCNM(dist(xy))

(PCNM$threshold) #Truncation distance

(length(PCNM$values)) #Number of eigenvalues

PCNM$Moran\_I

# selecte posive Moran I

select = which(PCNM$Moran\_I$Positive == T)

PCNM\_pos = as.data.frame(PCNM$vectors)[,select]

## Variables were used to partition variation in the beta-deviations into individual fractions explained by environmental and spatial variables

spa.trans <- scale(spa)

spa.trans <- as.data.frame(spa.trans)

env.trans <- scale(env)

env.trans <- as.data.frame(env.trans)

mod <- varpart(dev, env.trans, spa.trans)

mod

plot(mod, cutoff = 0, digits = 2)

## forward-model selection regression ('step' function in the R stats package) was performed to examine

## the relationship between explanatory variables and beta-deviation

spa.lm <- lm(dev$deviation ~ ., data =spa.trans)

step\_spa<-step(spa.lm, direction = "forward")

summary(step(step\_spa))

env.lm <- lm(dev$deviation ~ ., data =env.trans)

step\_env<-step(env.lm, direction = "forward")

summary(step(step\_env))